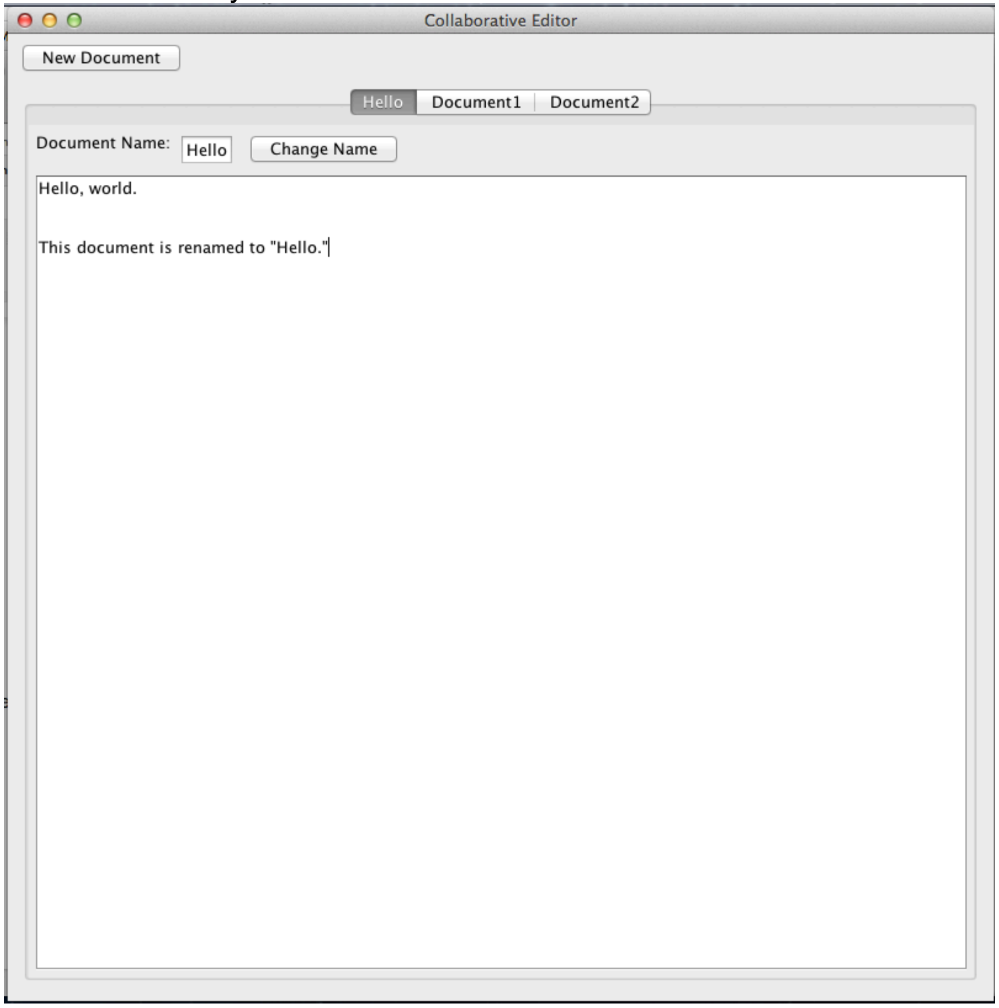
Collaborative Editor Design Documentation

angelaz-mturek-nedmonds

Overall Design:

Purpose: To design a collaborative editor that allows multiple users to work on a single document simultaneously across a network.

GUI: Built with the Java Swing library. Each user, when connected to the server, will be updated with all the documents currently stored in the server. When each user edits the document, their local changes will be displayed instantaneously on their own GUI. These changes will also be submitted to the server into the server's "edit queue." Once an item on the "edit queue" is processed by the server, the server sends an updated message to each client using a protocol described in the Communication Protocol section. The GUI contains: 1) a "New Document" button that creates a new document, 2) a "Change Name" button that renames the document, 3) several tabs, each containing a document, and 4) the text pane that contains the body of the document.



Hello, world.

This document is renamed to "Hello."

Backend:

Server: The server performs the following tasks: 1) maintains a queue of edits named “edit queue”, 2) processes each client’s edit request, placing them in order of the server receiving the edit requests from the clients, determines whether these requests are still valid, and accept valid requests as appropriate, 3) every time a valid request is accepted, sends an update request to the client, 4) keeps an auto-created copy of a client which does not post any update requests and receives all updates from other clients. NOTE: The server’s main job is to maintain and process the edit queue. It does not keep a “server” copy of the documents, instead, it keeps a copy of a “client” to back up changes.

Client: Each client 1) keeps a copy of the documents, 2) sends and receives update requests to/from the server.

Classes: The Model part of the Collaborative Editor has the following classes: 1) Anchor, 2) Document, 3) Node.

Edit: An edit by a user is defined to be one of the following: 1) insert, 2) delete, 3) block insert, 4) block delete.

Anchors: The server will store an “anchor” for each user. This is essentially just a pair of unique id’s which indicate where a user’s cursor is in the document. The first id corresponds to the node before the user’s cursor, while the second id is the node after the cursor. The reason that both are stored is to handle simultaneous insertions by users. If the server sees that a user’s anchor has nodes that are not consecutive, then it can tell that two user’s have been editing in the same location and will send messages back to the clients appropriately. The details are described in the Insertion method description

Documents: Each Document contains a String representing the Document Name, an int representing a unique Document ID, a LinkedList of Nodes representing the content of the each document, and an ArrayList of Anchors that holds all the Anchors for all the users.

Nodes: A node is an immutable object that holds the base units (characters, punctuation, white space, tab, newline character, etc) that make up the content of the Document. Each nodes has an int representing a unique ID the server assigns to each base unit and a String representing the actual base units.

Client Actions/Edits:

Insertions:

- **Single character insertions**: To handle insertions, we are assuming that all the users are holding valid anchors which specify between which 2 nodes their cursor is. Entering a character will then result in sending an “insert CHARACTER” message to the server - the message from the user does not (and should not) specify the actual index at which they are inserting, that would cause problems in concurrent editing since indices change based on other users’ actions. The server then evaluates whether the anchor has been “invalidated” by previous insertions (it no longer connects 2 consecutive nodes) in which case it has to be updated to produce the desired behavior as specified by the documentation. If the anchor is still valid, a new node is inserted and the position of the anchor is shifted to point to it.
- **Block insertions (copy-and-paste)**: In the server queue, a block insertion is just a sequence of single-character insertions. It cannot be sent as such from the client though since that would result in possible interleaving with other users’ requests. The client will therefore be able to send another type of message to the server that groups these insertions together as an atomic request. The server then decomposes it for the queue and distributes it to all the clients as simple insertions.

Deletions:

- **Single character deletions**: After the user has acquired an anchor in the text, pressing backspace / delete results in sending a message to the server with a deletion request for the immediately preceding / following node. The requests contain the unique ID of the node to be deleted. In that way, simultaneous deletion requests for the same node do not result in more nodes getting deleted than intended. (We rejected the idea that a person just sends a delete request relative to their anchor as those would produce undesirable behavior). All anchors connected to the deleted node will be shifted to the node immediately before.
- **Block deletions**: This is very similar to single character deletions except both the ID of the first and the last node to be deleted have to be sent to the server by the client. The server then loops through the LinkedList and shifts all the anchors connected to any of the nodes to the node that immediately precedes the beginning of the deleted block.

Protocols:

Most messages to the server will be structured in 3 parts.
1. “who” will be a string that identifies which user is sending the message to the server. For example, “u2” would indicate that user 2 is sending the message.

2. “what” will indicate which of the actions is being passed. The options are *insert, delete, block-delete block-insert, update-anchor.*
3. “content” will contain one of a few things depending on the “what”. If it is insert, it will contain the character to insert. A block insert will contain the string to insert. Single deletions will have the unique id being removed, block-deletions will have the start and end ids of the block being deleted. If it is an update anchor request, then it would contain the unique ids of the “start” and “end” locations of the anchor update.

There are a few “special” messages. new doc, update document name, and bye which are only sent from the client to the server unlike the other messages.

ClienttoServerRequest ::= (ServertoClientRequest | NewDoc | UpdateDocName | Bye)
ServertoClientRequest ::= (Insert | Delete | Block-Insert | Block-Delete | Anchor-Update)
Insert ::= (Who insert [\x00-\x7F])
Block-Insert ::= (Who block-insert [\x00-\x7F]+)
Delete ::= (Who delete [0-9]+)
Block-Delete ::= (Who block-delete [0-9]+ [0-9]+)
Anchor-Update ::= (Who update-anchor [0-9]+
Who::= u[0-9]+
NewDoc ::= newdoc
UpdateDocName ::= (docID, newName)
Bye ::= (Who bye)
docID ::= [\x00-\x7F]+
newName ::= [\x00-\x7F]+